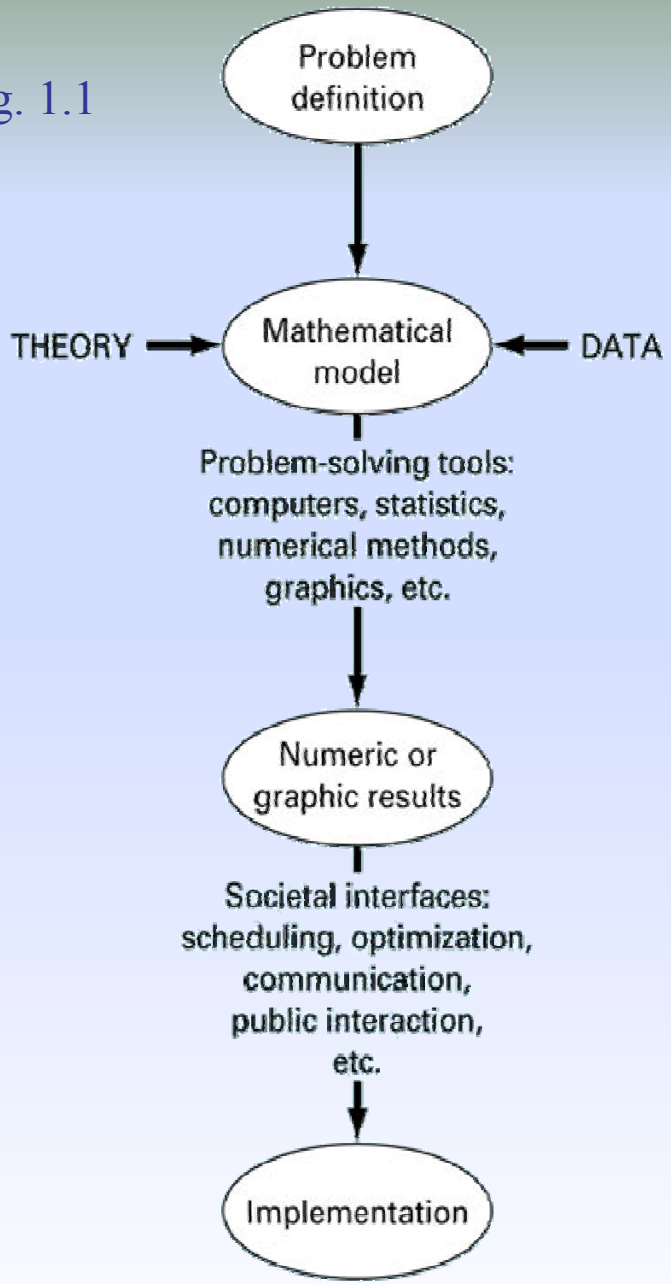
Mathematical Modeling and Engineering Problem solving

Chapter 1

- Requires understanding of engineering systems
  - –By observation and experiment
  - –Theoretical analysis and generalization
- Computers are great tools, however, without fundamental understanding of engineering problems, they will be useless.

Fig. 1.1


Problem definition
THEORY
Mathematical model
DATA
Problem-solving tools: computers, statistics, numerical methods, graphics, etc.
Numeric or graphic results
Societal interfaces: scheduling, optimization, communication, public interaction, etc.
Implementation

- A mathematical model is represented as a functional relationship of the form

$$\text{Dependent Variable} = f\left(\begin{array}{l}\text{independent variables, parameters,} \\ \text{forcing functions}\end{array}\right)$$

- *Dependent variable*: Characteristic that usually reflects the state of the system
- *Independent variables*: Dimensions such as time ans space along which the systems behavior is being determined
- *Parameters*: reflect the system's properties or composition
- *Forcing functions*: external influences acting upon the system

Newton's 2nd law of Motion

- States that "*the time rate change of momentum of a body is equal to the resulting force acting on it.*"
- The model is formulated as

$$\boldsymbol{F = m\,a}$$

F=net force acting on the body (N)

m=mass of the object (kg)

a=its acceleration (m/s^2)

- Formulation of Newton's 2nd law has several characteristics that are typical of mathematical models of the physical world:
  - It describes a natural process or system in mathematical terms
  - It represents an idealization and simplification of reality
  - Finally, it yields reproducible results, consequently, can be used for predictive purposes.

- Some mathematical models of physical phenomena may be much more complex.

- Complex models may not be solved exactly or require more sophisticated mathematical techniques than simple algebra for their solution

  – Example, modeling of a falling parachutist:

$$\frac{dv}{dt} = \frac{F}{m}$$

Force due to gravity

$$F = F_D + F_U$$ ← Upward force due to air resistance

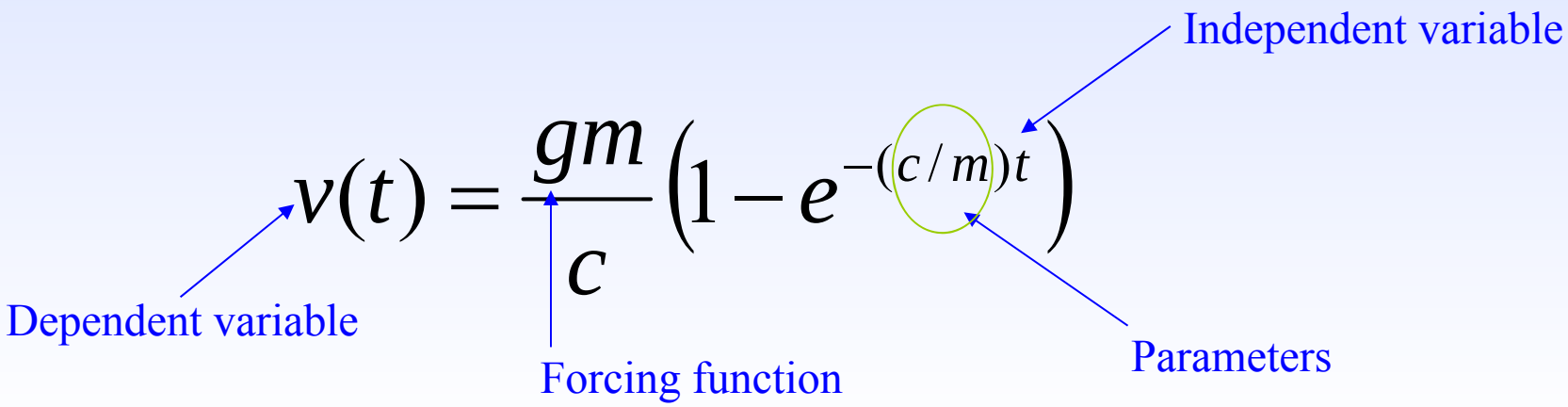
$$F_D = mg$$

$$F_U = -cv$$ c: a constant (drag coefficient in kg/s)

$$\frac{dv}{dt} = \frac{mg - cv}{m}$$

$$\frac{dv}{dt} = g - \frac{c}{m}v$$

- This is a differential equation and is written in terms of the differential rate of change dv/dt of the variable that we are interested in predicting.
- If the parachutist is initially at rest (v=0 at t=0), using calculus



EXAMPLE 1.1 Analytical Solution to the Falling Parachutist Problem

Problem Statement. A parachutist of mass 68.1 kg jumps out of a stationary hot air balloon. Use Eq. (1.10) to compute velocity prior to opening the chute. The drag coefficient is equal to 12.5 kg/s.

Solution. Inserting the parameters into Eq. (1.10) yields

$$v(t) = \frac{9.81(68.1)}{12.5}(1 - e^{-(12.5/68.1)t}) = 53.44\,(1 - e^{-0.18355t})$$

which can be used to compute

t, s	v, m/s
0	0.00
2	16.42
4	27.80
6	35.68
8	41.14
10	44.92
12	47.54
∞	53.44

According to the model, the parachutist accelerates rapidly (Fig. 1.3). A velocity of 44.92 m/s is attained after 10 s. Note also that after a sufficiently long time, a constant velocity, called the *terminal velocity,* of 53.44 m/s is reached. This velocity is constant because, eventually, the force of gravity will be in balance with the air resistance. Thus, the net force is zero and acceleration has ceased.

Programming and Software

Chapter 2

- Objective is how to use the computer as a tool to obtain numerical solutions to a given engineering model. There are two ways in using computers:
  - Use available software
  - Or, write computer programs to extend the capabilities of available software, such as Excel and Matlab.
- Engineers should not be tool limited, it is important that they should be able to do both!

- Computer programs are set of instructions that direct the computer to perform a certain task.
- To be able to perform engineering-oriented numerical calculations, you should be familiar with the following programming topics:
  - Simple information representation (constants, variables, and type declaration)
  - Advanced information representation (data structure, arrays, and records)
  - Mathematical formulas (assignment, priority rules, and intrinsic functions)
  - Input/Output
  - Logical representation (sequence, selection, and repetition)
  - Modular programming (functions and subroutines)
- We will focus the last two topics, assuming that you have some prior exposure to programming.

Structured Programming

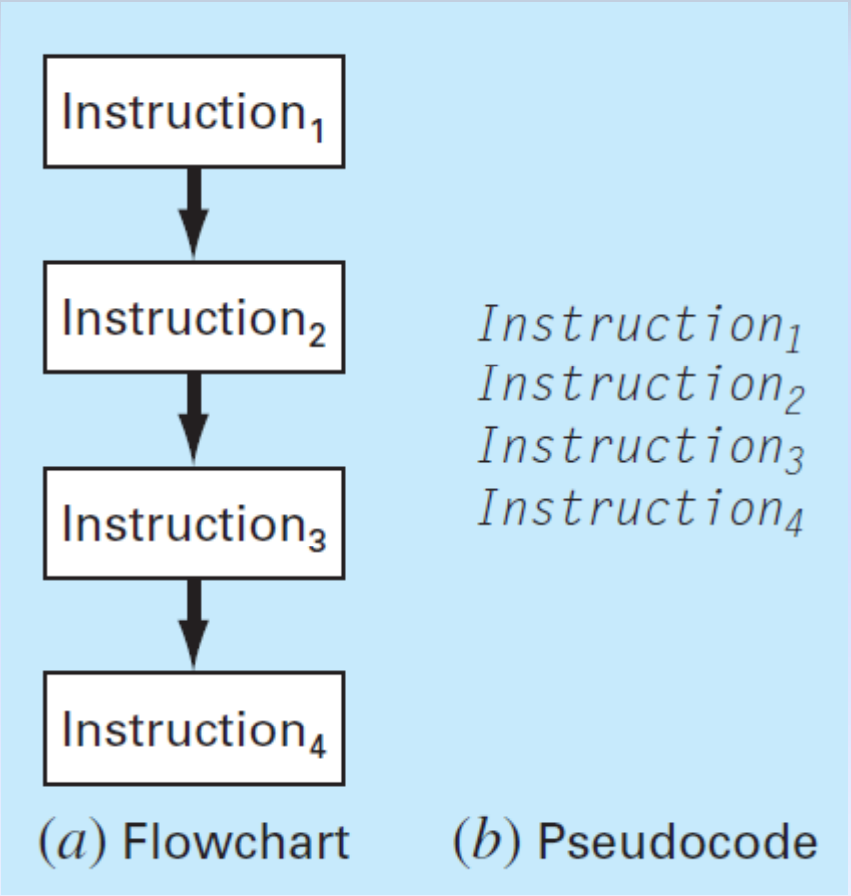
- *Structured programming* is a set of rules that prescribe god style habits for programmer.
  - An organized, well structured code
  - Easily sharable
  - Easy to debug and test
  - Requires shorter time to develop, test, and update
- The key idea is that any numerical algorithm can be composed of using the three fundamental structures:
  - Sequence, selection, and repetition

FIGURE 2.1
Symbols used in flowcharts.

SYMBOL	NAME	FUNCTION
	Terminal	Represents the beginning or end of a program.
	Flowlines	Represents the flow of logic. The humps on the horizontal arrow indicate that it passes over and does not connect with the vertical flowlines.
	Process	Represents calculations or data manipulations.
	Input/output	Represents inputs or outputs of data and information.
	Decision	Represents a comparison, question, or decision that determines alternative paths to be followed.
	Junction	Represents the confluence of flowlines.
	Off-page connector	Represents a break that is continued on another page.
	Count-controlled loop	Used for loops which repeat a prespecified number of iterations.

- Sequence. Computer code must be implemented one instruction at a time, unless you instruct otherwise. The structure can be expressed as a flowchart or pseudocode.

Fig.2.2



Pseudocode is an informal high-level description of the operating principle of a computer program or other algorithm.

- Selection.

  Splits the program's flow into branches based on outcome of a logical condition.

  One or a number of statements is executed depending on the state of the program. This is usually expressed with keywords such as if..then..else..endif

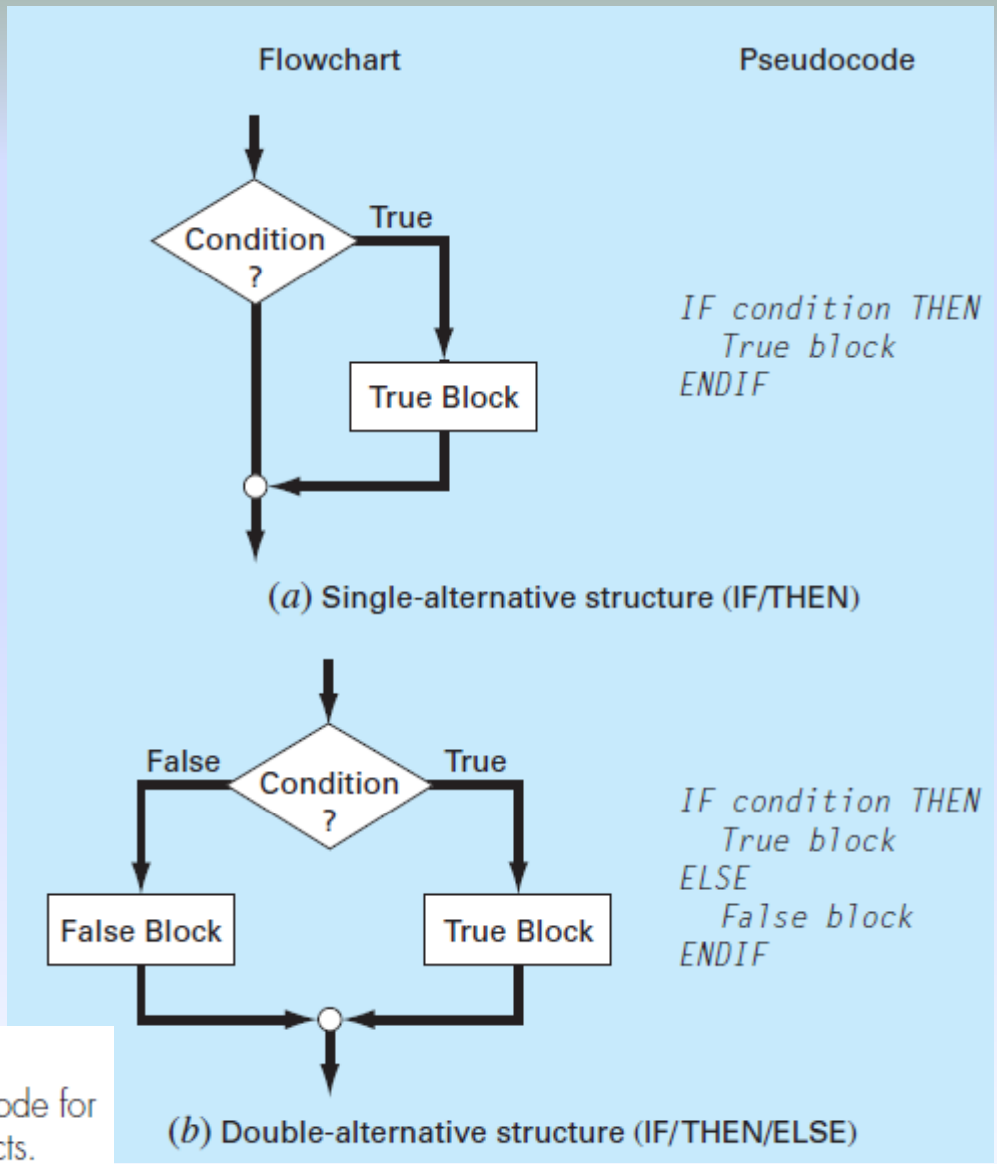
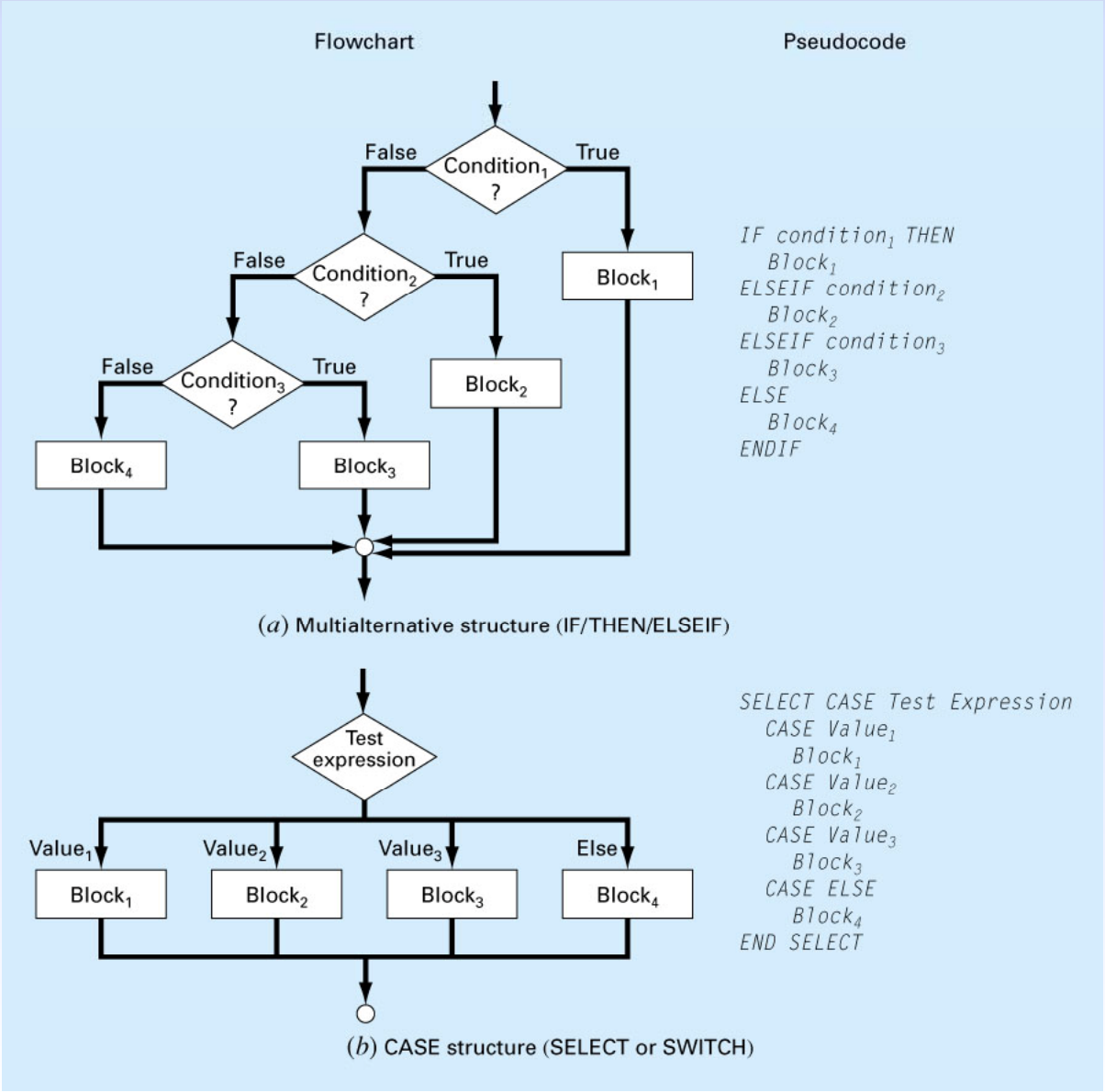


FIGURE 2.3
Flowchart and pseudocode for simple selection constructs. (a) Single-alternative selection (IF/THEN) and (b) double-alternative selection (IF/THEN/ELSE).

- Repetition (*or* iteration) means to implement instructions repeatedly.

FIGURE 2.4
Flowchart and pseudocode for supplementary selection or branching constructs. (a) Multiple-alternative selection (IF/THEN/ELSEIF) and (b) CASE construct.



A simple loop

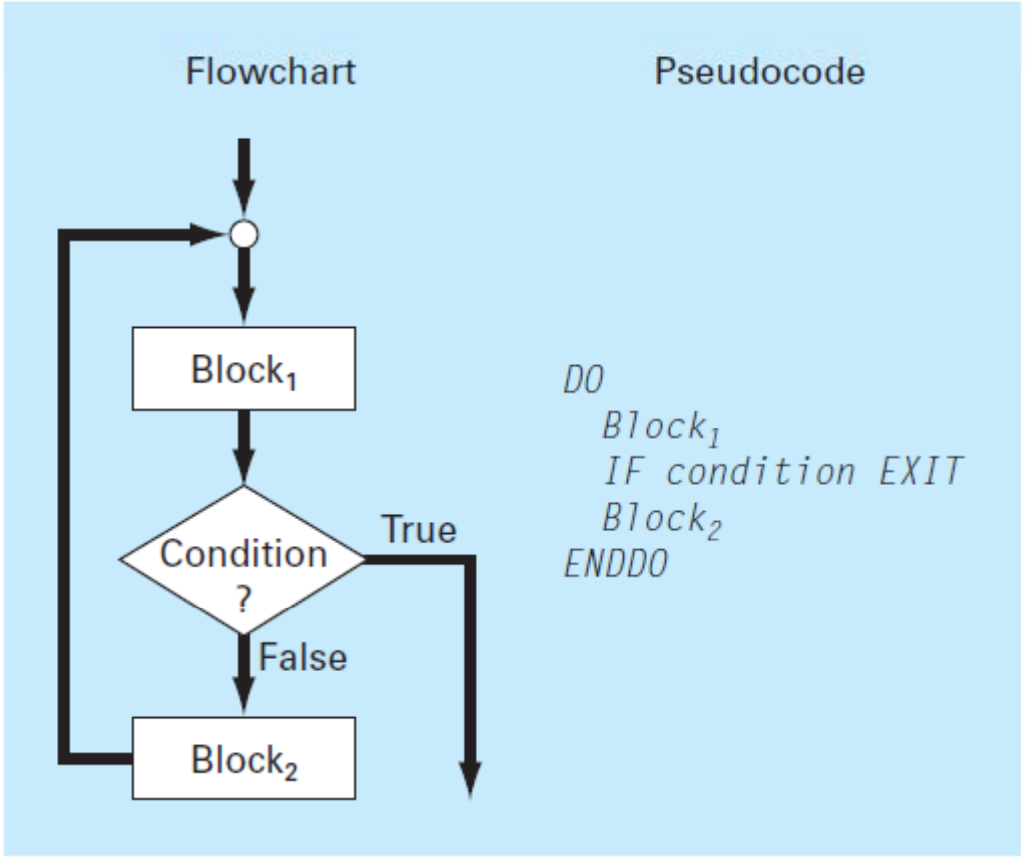


FIGURE 2.5
The DOEXIT or break loop.

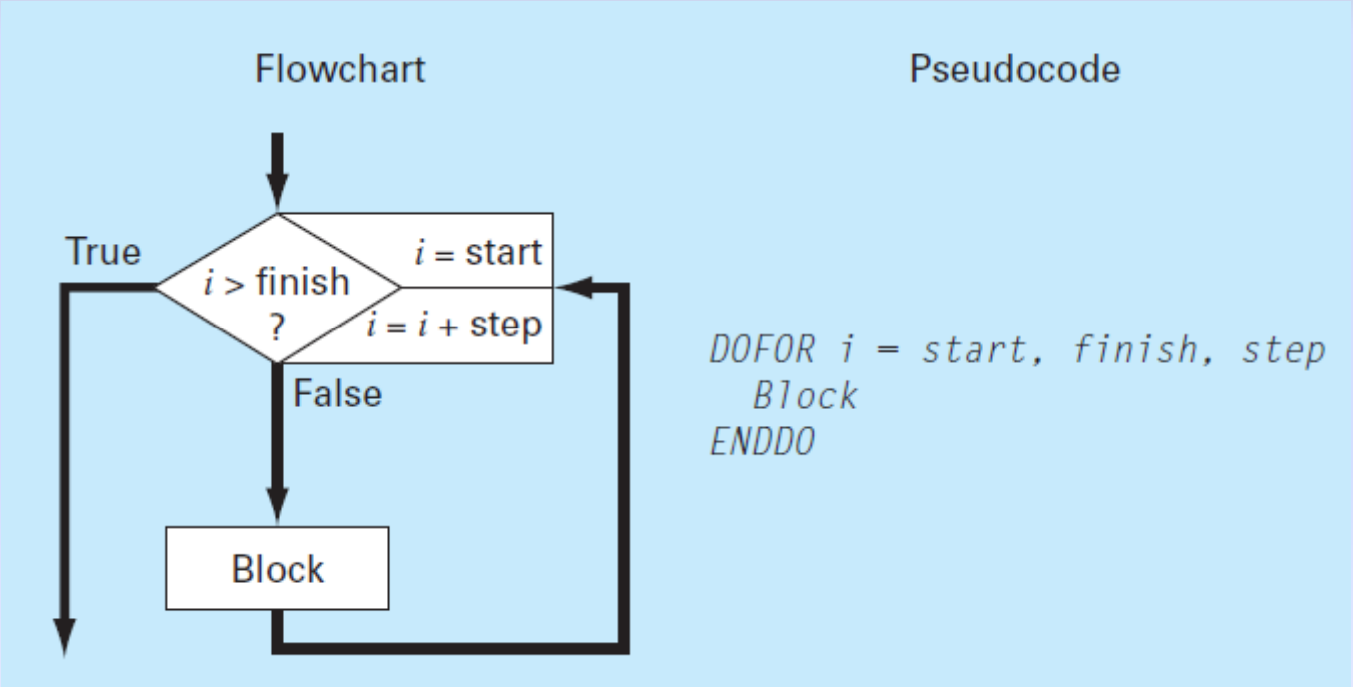


FIGURE 2.6
The count-controlled or DOFOR construct.

Iterations are usually expressed with keywords such as while, repeat, for or do..until

Modular Programming

- The computer programs can be divided into *subprograms*, or *modules*, that can be developed and tested separately.
- Modules should be as independent and self contained as possible.
- Advantages to modular design are:
  - It is easier to understand the underlying logic of smaller modules
  - They are easier to debug and test
  - Facilitate program maintenance and modification
  - Allow you to maintain your own library of modules for later use

An example pseudocode suitable for modular programming

FIGURE 2.7
Pseudocode for a function that solves a differential equation using Euler's method.

```
FUNCTION Euler(dt, ti, tf, yi)
t = ti
y = yi
h = dt
DO
  IF t + dt > tf THEN
    h = tf - t
  ENDIF
  dydt = dy(t, y)
  y = y + dydt * h
  t = t + h
  IF t ≥ tf EXIT
ENDDO
Euler = y
END Euler
```

EXCEL

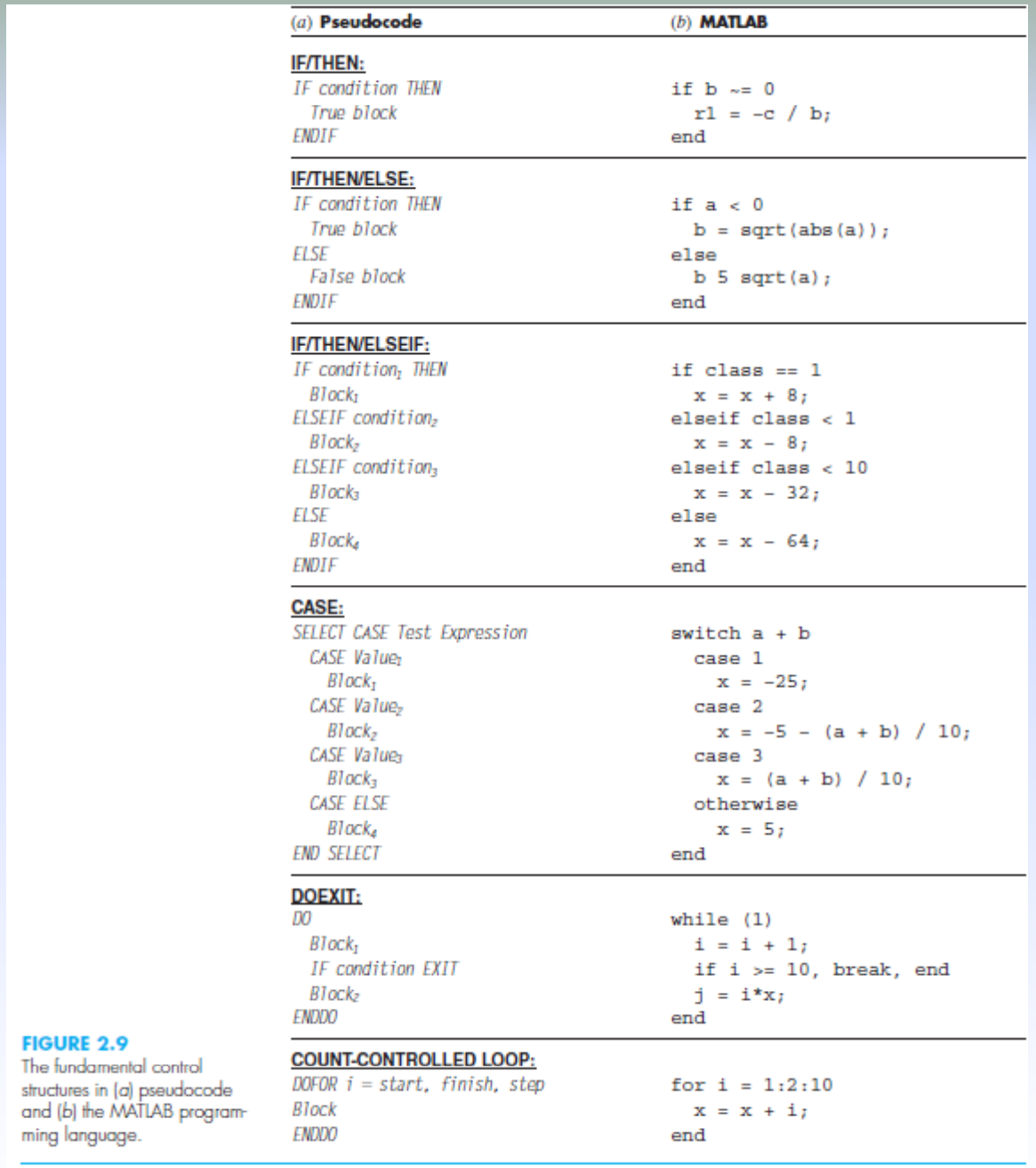
- Is a spreadsheet that allow the user to enter and perform calculations on rows and columns of data.
- When any value on the sheet is changed, entire calculation is updated, therefore, spreadsheets are ideal for “what if?” sorts of analysis.
- Excel has some built in numerical capabilities including equation solving, curve fitting and optimization.
- It also includes VBA as a macro language that can be used to implement numerical calculations.
- It has several visualization tools, such as graphs and three dimensional plots.

(a) Pseudocode	(b) Excel VBA
IF/THEN:	
IF condition THEN	`If b <> 0 Then`
True block	`  r1 = -c / b`
ENDIF	`End If`
IF/THEN/ELSE:	
IF condition THEN	`If a < 0 Then`
True block	`  b = Sqr(Abs(a))`
ELSE	`Else`
False block	`  b = Sqr(a)`
ENDIF	`End If`
IF/THEN/ELSEIF:	
IF condition$_1$ THEN	`If class = 1 Then`
Block$_1$	`  x = x + 8`
ELSEIF condition$_2$	`ElseIf class < 1 Then`
Block$_2$	`  x = x - 8`
ELSEIF condition$_3$	`ElseIf class < 10 Then`
Block$_3$	`  x = x - 32`
ELSE	`Else`
Block$_4$	`  x = x - 64`
ENDIF	`End If`
CASE:	
SELECT CASE Test Expression	`Select Case a + b`
CASE Value$_1$	`  Case Is < -50`
Block$_1$	`    x = -5`
CASE Value$_2$	`  Case Is < 0`
Block$_2$	`    x = -5 - (a + b) / 10`
CASE Value$_3$	`  Case Is < 50`
Block$_3$	`    x = (a + b) / 10`
CASE ELSE	`  Case Else`
Block$_4$	`    x = 5`
END SELECT	`End Select`
DOEXIT:	
DO	`Do`
Block$_1$	`  i = i + 1`
IF condition EXIT	`  If i >= 10 Then Exit Do`
Block$_2$	`  j = i*x`
ENDDO	`Loop`
COUNT-CONTROLLED LOOP:	
DOFOR i = start, finish, step	`For i = 1 To 10 Step 2`
Block	`  x = x + i`
ENDDO	`Next i`

FIGURE 2.8
The fundamental control structures in (a) pseudocode and (b) Excel VBA.

MATLAB

- Is a flagship software which was originally developed as a *MATrix LABoratory*. A variety of numerical functions, symbolic computations, and visualization tools have been added to the matrix manipulations.
- MATLAB is closely related to programming.



(*a*) **Pseudocode**	(*b*) **MATLAB**
IF/THEN:	
IF condition THEN	`if b ~= 0`
True block	`  r1 = -c / b;`
ENDIF	`end`
IF/THEN/ELSE:	
IF condition THEN	`if a < 0`
True block	`  b = sqrt(abs(a));`
ELSE	`else`
False block	`  b 5 sqrt(a);`
ENDIF	`end`
IF/THEN/ELSEIF:	
IF condition$_1$ THEN	`if class == 1`
Block$_1$	`  x = x + 8;`
ELSEIF condition$_2$	`elseif class < 1`
Block$_2$	`  x = x - 8;`
ELSEIF condition$_3$	`elseif class < 10`
Block$_3$	`  x = x - 32;`
ELSE	`else`
Block$_4$	`  x = x - 64;`
ENDIF	`end`
CASE:	
SELECT CASE Test Expression	`switch a + b`
CASE Value$_1$	`  case 1`
Block$_1$	`    x = -25;`
CASE Value$_2$	`  case 2`
Block$_2$	`    x = -5 - (a + b) / 10;`
CASE Value$_3$	`  case 3`
Block$_3$	`    x = (a + b) / 10;`
CASE ELSE	`  otherwise`
Block$_4$	`    x = 5;`
END SELECT	`end`
DOEXIT:	
DO	`while (1)`
Block$_1$	`  i = i + 1;`
IF condition EXIT	`  if i >= 10, break, end`
Block$_2$	`  j = i*x;`
ENDDO	`end`
COUNT-CONTROLLED LOOP:	
DOFOR i = start, finish, step	`for i = 1:2:10`
Block	`  x = x + i;`
ENDDO	`end`

FIGURE 2.9
The fundamental control structures in (a) pseudocode and (b) the MATLAB programming language.

Other Languages and Libraries

- Maple
- Mathcad
- Fortran 90 (IMSL)
- C++
- Visual basic etc…

<end of Chapter 2>

Approximations and Round-Off Errors

Chapter 3

- For many engineering problems, we cannot obtain analytical solutions.
- Numerical methods yield approximate results, results that are close to the exact analytical solution. We cannot exactly compute the errors associated with numerical methods.
  - Only rarely given data are exact, since they originate from measurements. Therefore there is probably error in the input information.
  - Algorithm itself usually introduces errors as well, e.g., unavoidable round-offs, etc …
  - The output information will then contain error from both of these sources.
- How confident we are in our approximate result?
- The question is "***how much error is present in our calculation and is it tolerable?***"

- Accuracy. How close is a computed or measured value to the true value
- Precision (or *reproducibility*). How close is a computed or measured value to previously computed or measured values.
- Inaccuracy (or *bias*). A systematic deviation from the actual value.
- Imprecision (or *uncertainty*). Magnitude of scatter.

Fig. 3.2

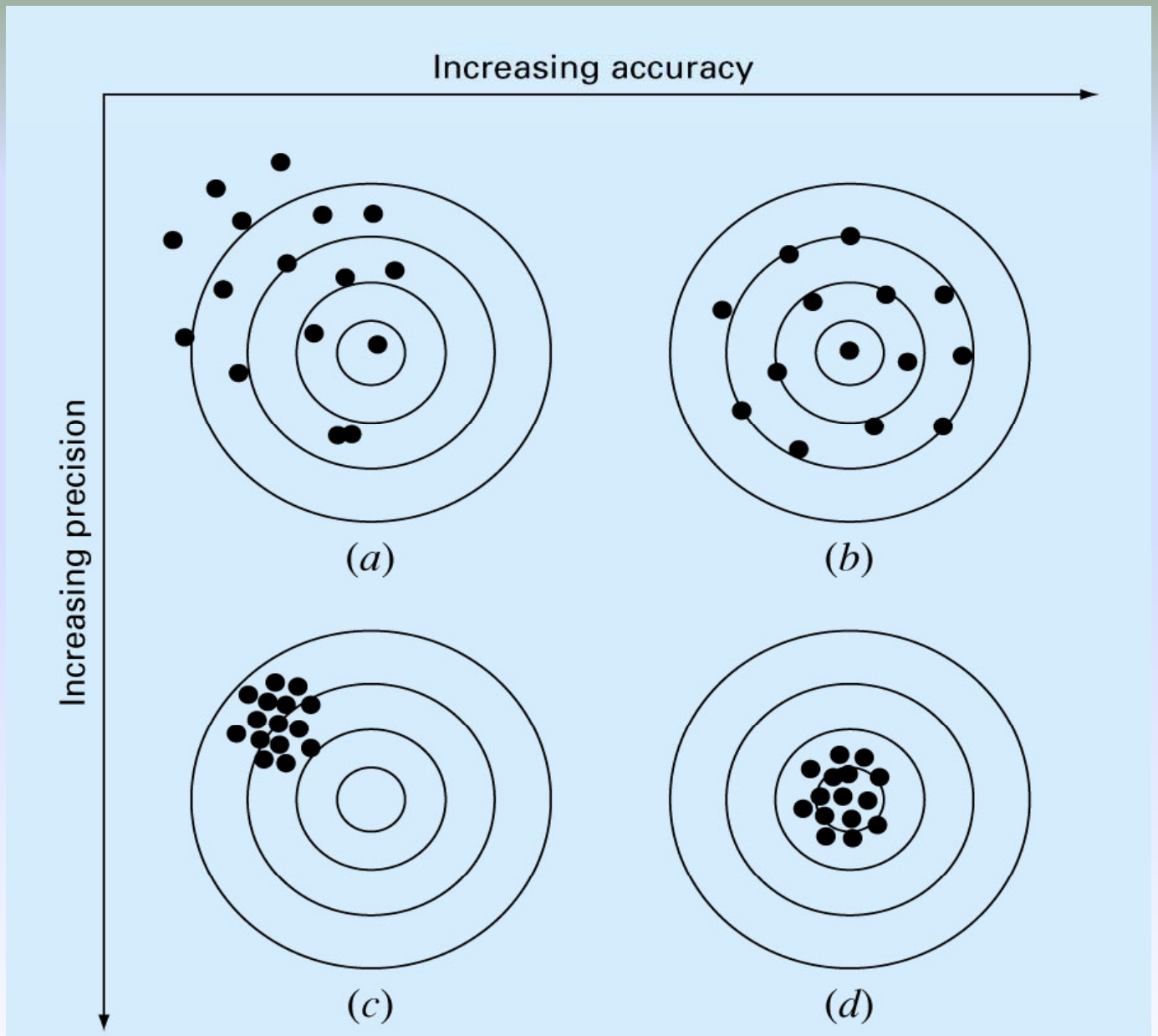


FIGURE 3.2
An example from marksmanship illustrating the concepts of accuracy and precision. (*a*) Inaccurate and imprecise; (*b*) accurate and imprecise; (*c*) inaccurate and precise; (*d*) accurate and precise.

Significant Figures

- Number of significant figures indicates precision. Significant digits of a number are those that can be *used* with *confidence,* e.g., the number of certain digits plus one estimated digit.

53.800 How many significant figures?

5.38×10^4	3
5.380×10^4	4
5.3800×10^4	5

Zeros are sometimes used to locate the decimal point not significant figures.

0.00001753	4
0.0001753	4
0.001753	4

Error Definitions

True Value = Approximation + Error

E_t = True value – Approximation (+/-)

True error

$$\text{True fractional relative error} = \frac{\text{true error}}{\text{true value}}$$

$$\text{True percent relative error, } \varepsilon_t = \frac{\text{true error}}{\text{true value}} \times 100\%$$

EXAMPLE 3.1 Calculation of Errors

Problem Statement. Suppose that you have the task of measuring the lengths of a bridge and a rivet and come up with 9999 and 9 cm, respectively. If the true values are 10,000 and 10 cm, respectively, compute (*a*) the true error and (*b*) the true percent relative error for each case.

Solution.

(a) The error for measuring the bridge is [Eq. (3.2)]

$$E_t = 10{,}000 - 9999 = 1 \text{ cm}$$

and for the rivet it is

$$E_t = 10 - 9 = 1 \text{ cm}$$

(b) The percent relative error for the bridge is [Eq. (3.3)]

$$\varepsilon_t = \frac{1}{10{,}000}100\% = 0.01\%$$

and for the rivet it is

$$\varepsilon_t = \frac{1}{10}100\% = 10\%$$

Thus, although both measurements have an error of 1 cm, the relative error for the rivet is much greater. We would conclude that we have done an adequate job of measuring the bridge, whereas our estimate for the rivet leaves something to be desired.

- For numerical methods, the true value will be known only when we deal with functions that can be solved analytically (simple systems). In real world applications, we usually not know the answer a priori. Then

$$\varepsilon_a = \frac{\text{Approximate error}}{\text{Approximation}} \times 100\%$$

- *Iterative approach*, example Newton's method

$$\varepsilon_a = \frac{\text{Current approximation - Previous approximation}}{\text{Current approximation}} \times 100\%$$

(+ / -)

- We can use absolute value of the error for termination.
- Computations are repeated until stopping criterion is satisfied.

$$|\varepsilon_a| < \varepsilon_s$$

Pre-specified % tolerance based on the knowledge of your solution

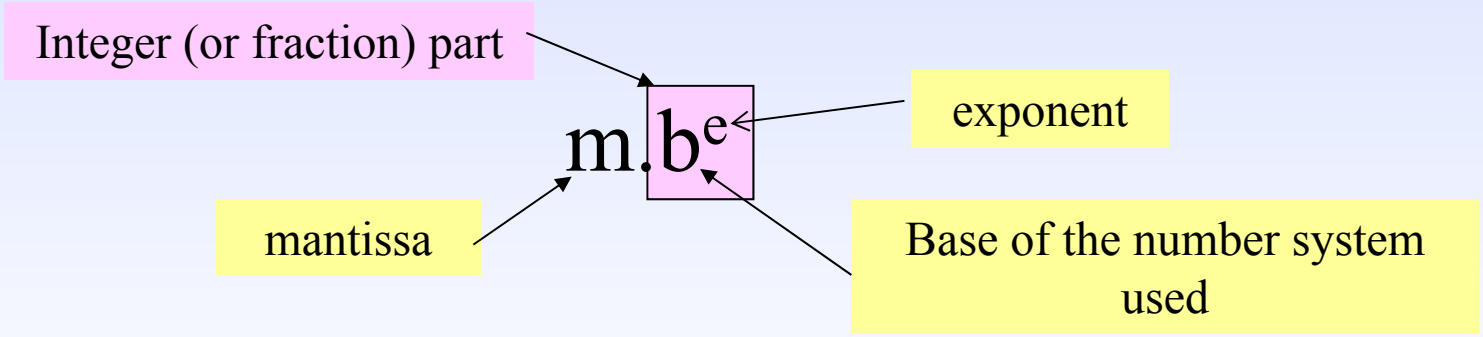
- If the following criterion is met

$$\varepsilon_s = (0.5 \times 10^{(2-n)})\%$$

you can be sure that the result is correct to at least *n* significant figures.

Round-off Errors

- Numbers such as π, e, or $\sqrt{7}$ cannot be expressed by a fixed number of significant figures.
- Computers use a base-2 representation, they cannot precisely represent certain exact base-10 numbers.
- Fractional quantities are typically represented in computer using "floating point" form, e.g.,



Number systems: positional notation

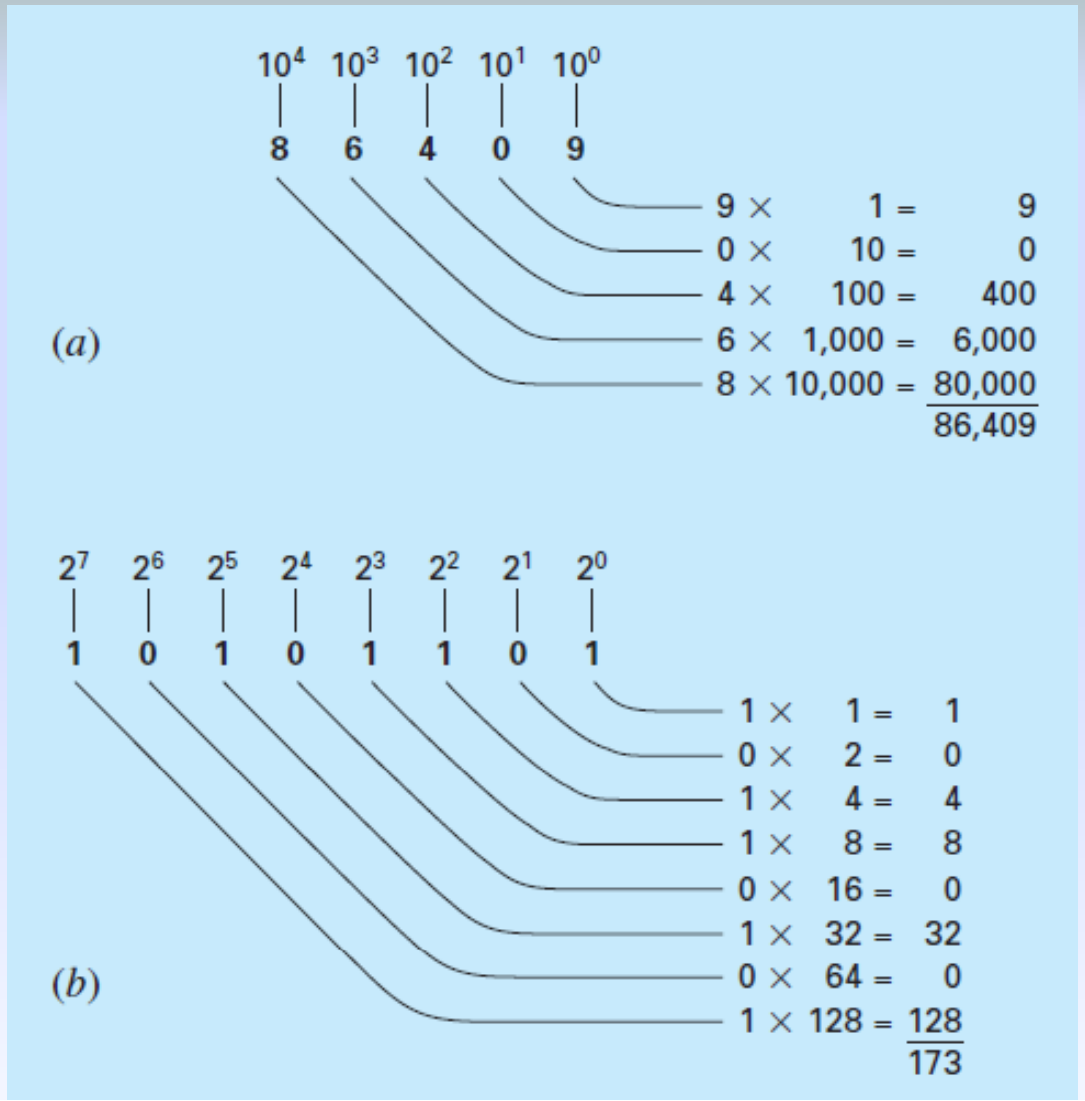


FIGURE 3.5
How the (*a*) decimal (base-10) and the (*b*) binary (base-2) systems work. In (*b*), the binary number 10101101 is equivalent to the decimal number 173.

Integer Representation. Now that we have reviewed how base-10 numbers can be represented in binary form, it is simple to conceive of how integers are represented on a computer. The most straightforward approach, called the signed magnitude method, employs the first bit of a word to indicate the sign, with a 0 for positive and a 1 for negative. The remaining bits are used to store the number. For example, the integer value of −173 would be stored on a 16-bit computer, as in Fig. 3.6.

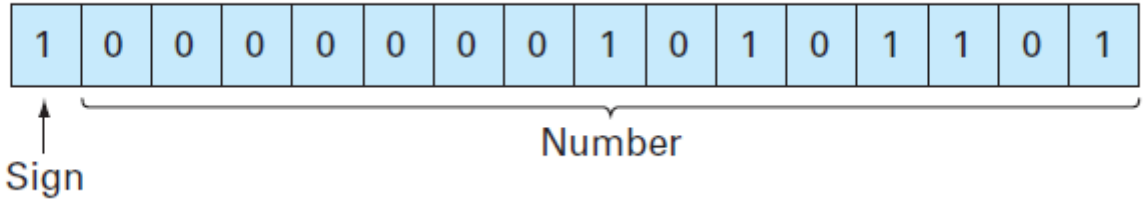
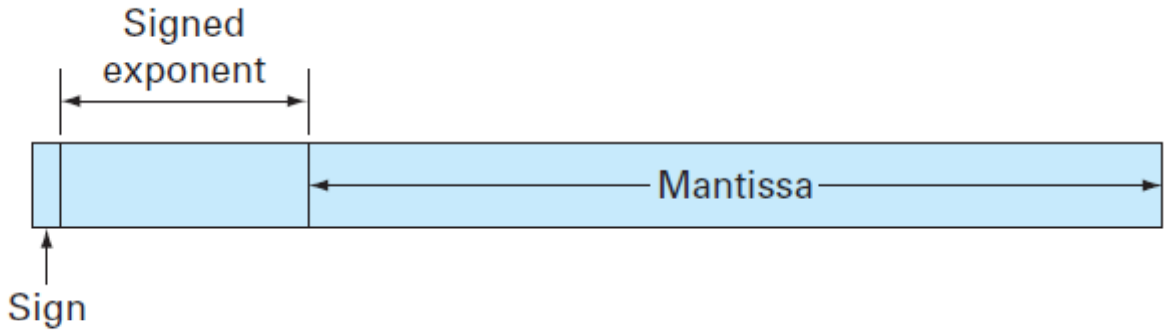


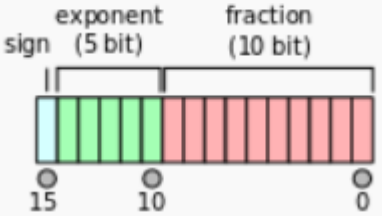
FIGURE 3.6
The representation of the decimal integer −173 on a 16-bit computer using the signed magnitude method.

Floating Point Representation.

FIGURE 3.7
The manner in which a floating-point number is stored in a word.

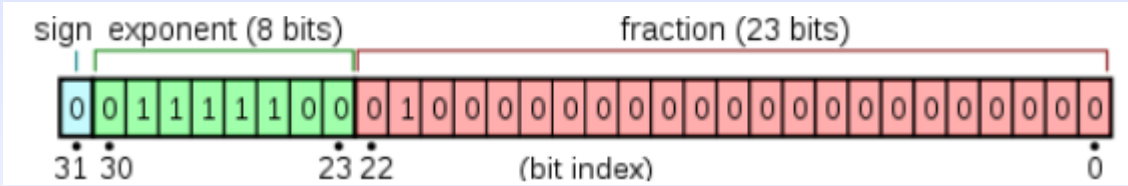


Floating-point precisions



IEEE 754

16-bit: Half (binary16)
32-bit: **Single** (binary32), decimal32
64-bit: Double (binary64), decimal64
128-bit: Quadruple (binary128), decimal128



32-bit: Single
(higher precision than 16-bit case)

156.78 ⏩ 0.15678×10^3 in a floating point base-10 system

$$\frac{1}{34} = 0.029411765$$

0.0294×10^0

Suppose only 4 decimal places to be stored

- Normalized to remove the leading zeroes. Multiply the mantissa by 10 and lower the exponent by 1

0.2941×10^{-1}

Additional significant figure is retained. This representation is a better representation than the previous case.

$$\frac{1}{b} \le |m| < 1$$ → Best choice for the mantissa range

Therefore

for a base-10 system $0.1 \le m < 1$

for a base-2 system $0.5 \le m < 1$

- Floating point representation allows both fractions and very large numbers to be expressed on the computer. However,
  - Floating point numbers take up more room.
  - Take longer to process than integer numbers.
  - Round-off errors are introduced because mantissa holds only a finite number of significant figures.

Chopping or Rounding?

Example:

π=3.14159265358 to be stored on a base-10 system carrying 7 significant digits.

π=3.141592 chopping error ε_t=0.00000065

If rounded

π=3.141593 ε_t=0.00000035

- Some machines use chopping, because rounding adds to the computational overhead. Since number of significant figures is large enough, resulting chopping error is negligible.